EE 334

SEGMENT OVERRIDE PREFIX 5

The **segment override prefix,** which may be added to almost any instruction in any memory-addressing mode, allows the programmer to deviate from the default segment. The segment override prefix is an additional byte that appends the front of an instruction to select an alternate segment register. About the only instructions that cannot be prefixed are the jump and call instructions that must use the code segment register for address generation.

For example, the MOV AX,[DI] instruction accesses data within the data segment by default. If required by a program, this can be changed by prefixing the instruction. Suppose that the data are in the extra segment instead of the data segment. This instruction addresses the extra segment if changed to MOV AX,ES:[DI].

The following table show some altered instructions that address different memory segments than normal

Assembly Language	*Segment Accessed*	*Default Segment*
MOV AX,DS:[BP]	Data	Stack
MOV AX,ES:[BP]	Extra	Stack
MOV AX,SS:[DI]	Stack	Data
MOV AX,CS:LIST	Code	Data
MOV AX,ES:[SI]	Extra	Data

Arithmetic Instructions

- The arithmetic instructions include
  - Addition
  - Subtraction
  - Multiplication
  - Division
- Data formats
  - Unsigned binary bytes
  - Signed binary bytes
  - Unsigned binary words
  - Signed binary words
  - Unpacked decimal bytes
  - Packed decimal bytes
  - ASCII numbers

Intel 8086 has 20 instructions for performing integer addition, subtraction, multiplication, division, and conversions from binary coded decimal to binary.

Addition	ADD d,s	; add byte or word
	ADC d,s	; add byte or word with carry
	INC d	; increment byte or word by adding 1
	DAA	; decimal adjust after addition
	AAA	; ASCII adjust for addition

Subtraction	SUB d,s	; subtract, d-s →d
	SBB d,s	; subtract with borrow
	DEC d	; decrement byte or word (subtract 1)
	DAS	; decimal adjust after subtraction
	AAS	; ASCII adjust for subtraction

Multiply
- MUL s ; multiply byte or word, unsigned
- IMUL s ; integer multiply, signed
- AAM ; ASCII adjust for multiply

Division
- DIV s ; unsigned byte or word divide
- IDIV s ; signed byte or word divide
- AAD ; ASCII adjust for divide

Other
- NEG d ; Negate, i.e. multiply by -1
- CBW ; convert byte to word and sign extend
- CWD ; convert word to double word
- CMP d, s ; compare byte or word

⌘ Signed integers may be represented simply by the use of 2's complement binary system, (hexadecimal is for human readability):

Examples (For 16-bit words)

Decimal	Binary	Hex
1	0000 0000 0000 0001	0001
-1	1111 1111 1111 1111	FFFF
29	0000 0000 0001 1101	001D
-29	1111 1111 1110 0011	FFE3
32,767	0111 1111 1111 1111	7FFF
-32,768	1000 0000 0000 0000	8000

4 bits = 1 nibble

8 bits = 1 byte

16 bits = 1 word (16 bit machines)

32 bits = 1 double-word

64 bits = 1 Quad-word

Binary Addition

- Can add immediate data to a register or memory.
- Can add data from a register to a register.
- Can add data from a register to memory.
- Can add data from memory to a register.
- Can NOT add data directly from one memory location to another.

Recall Full Adder Truth Table

C_i	A_i	B_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

C	1	1	1	0
A	0	1	0	1
B	0	1	1	1
	1	1	0	0

Final carry = 0

Binary Addition

Dec	Hex
53	35
+25	+19
78	4E

Binary

0	1	1	0	0	0	1	
0	0	1	1	0	1	0	1
0	0	0	1	1	0	0	1
0	1	0	0	1	1	1	0

Carry and Overflow

Dec	Hex
53	35
+25	+19
78	4E

C = 0
O = 0

Binary

0	1	1	0	0	0	1	
0	0	1	1	0	1	0	1
0	0	0	1	1	0	0	1
0	1	0	0	1	1	1	0

Note no carry from bit 6 to bit 7 and no carry from bit 7 to C.

Carry and Overflow

Dec	Hex		Binary
			1 1 1 1 1 1 1
53	35		0 0 1 1 0 1 0 1
+91	+5B		0 1 0 1 1 0 1 1
144	90	C = 0	1 0 0 1 0 0 0 0
		O = 1	

Note carry from bit 6 to bit 7 but no carry from bit 7 to C.

Thinking SIGNED we added two positive numbers and got a negative result. This can't be correct! Therefore, the OVERFLOW bit, O, is set to 1. Correct answer (144) is outside the range -128 to +127.

Carry and Overflow

Dec	Hex		Binary
			1 1 1 0 1 1 1
53	35		0 0 1 1 0 1 0 1
-45	+D3		1 1 0 1 0 0 1 1
8	108	C = 1	0 0 0 0 1 0 0 0
		O = 0	

Ignore carry

Note carry from bit 6 to bit 7 and carry from bit 7 to C.

Thinking SIGNED we added a positive number to a negative number and got the correct positive answer. Therefore, the OVERFLOW bit, O, is cleared to 0. Correct answer (8) is inside the range -128 to +127.

Carry and Overflow

Dec	Hex		Binary
			0 0 1 1 1 1 0
- 98	9E		1 0 0 1 1 1 1 0
- 45	+D3		1 1 0 1 0 0 1 1
- 143	171	C = 1	0 1 1 1 0 0 0 1
		O = 1	

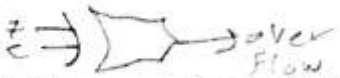
Ignore carry

Note no carry from bit 6 to bit 7 but there is a carry from bit 7 to C.

Thinking SIGNED we added two negative numbers and got a positive answer. This must be wrong! Therefore, the OVERFLOW bit, O, is set to 1. Correct answer (-143) is outside the range -128 to +127

Overflow

- **Note that the overflow bit was set whenever we had a carry from bit 6 to bit 7, but no carry from bit 7 to C.**
- **It was also set when we had a carry from bit 7 to C, but no carry from bit 6 to bit 7.**
- **Upshot: the overflow bit is the EXCLUSIVE-OR of a carry from bit 6 to bit 7 and a carry from bit 7 to C.**



Binary Code Decimal (BCD)

- Code decimal numbers using the binary digits, 0 - 9. That is, 0000 - 1001.
- Can NOT use the hex digits A - F. For example, the DECIMAL number 3582 would be coded in BCD as 0011 0101 1000 0010
- While this looks like the HEX number 3582H in BCD we interpret it as the DECIMAL number 3582.

BCD Arithmetic

- BCD: **B**inary **C**oded **D**ecimal
- **Packed BCD Arithmetic**
  - BCD Addition — Instruction : **DAA**
  - BCD Subtraction — Instruction : **DAS**
- **Unpacked BCD Arithmetic**
  - ASCII Addition — Instruction : **AAA**
  - ASCII Subtracition — Instruction : **AAS**
  - BCD Multiplication — Instruction : **AAM**
  - BCD Division — Instruction : **AAD**

Addition Instructions: ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	ADD D, S	(S) +(D)→(D) Carry →(CF)	OF, SF, ZF, AF, PF, CF
ADC	Add with carry	ADC D, S	(S) +(D)+(CF)→(D) Carry →(CF)	OF, SF, ZF, AF, PF, CF
INC	Increment by 1	INC D	(D) +1→(D)	OF, SF, ZF, AF, PF
AAA	ASCII adjust for addition	AAA		AF, CF OF, SF, ZF, PF undefined
DAA	Decimal adjust for addition	DAA		SF, ZF, AF, PF, CF, OF undefined

- **ADD Des, Src**
- **It adds a byte to byte or a word to word.**
- **It effects AF, CF, OF, PF, SF, ZF flags.**

- **Example:**

```
ADD AL, 7AH   ; adds 7AH to AL register
ADD DX, AX    ; adds AX to DX register
ADD AX, [BX]  ; adds [BX] to AX register
```

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

Allowed operands for ADD and ADC instructions

Register Addition

- Add the content of several registers.
- When arithmetic instructions executed, contents of the flag register change.
- Interrupt, trap, and other flags do not change.
- Any ADD instruction modifies the contents of the sign, zero, carry, auxiliary carry, parity, and overflow flags.
- Example:

```
ADD AX, BX    ; adds BX to AX register
ADD AX, CX    ; adds CX to AX register
ADD AX, DX    ; adds DX to AX register
```

Example

```
                    ; Addition of Un Signed numbers
ADD   CL, BL  ;CL = 01110011 = 115 decimal
              ;+ BL = 01001111 = 79 decimal
              ;Result in CL = 11000010 = 194 decimal
              ; Addition of Signed numbers
ADD   CL, BL  ;CL = 01110011 = + 115 decimal
              ;+ BL = 01001111 = +79 decimal
              Result in CL = 11000010 = - 62 decimal
; Incorrect because result is too large to fit in 7 bits.
```

Immediate Addition

- Immediate addition is employed whenever constant or known data are added.
- Example:

```
MOV DL, 12H
ADD DL, 33H
```

- The sum 45H is stored in DL register.
- Flags changes, as follows:
- Z = 0 (result not zero), S = 0 (result positive), C = 0 (no carry), P = 0 (odd parity), AC = 0 (no half carry), O = 0 (no overflow).

- **Memory-to-Register Addition**
- **Moves memory data to be added to a register.**

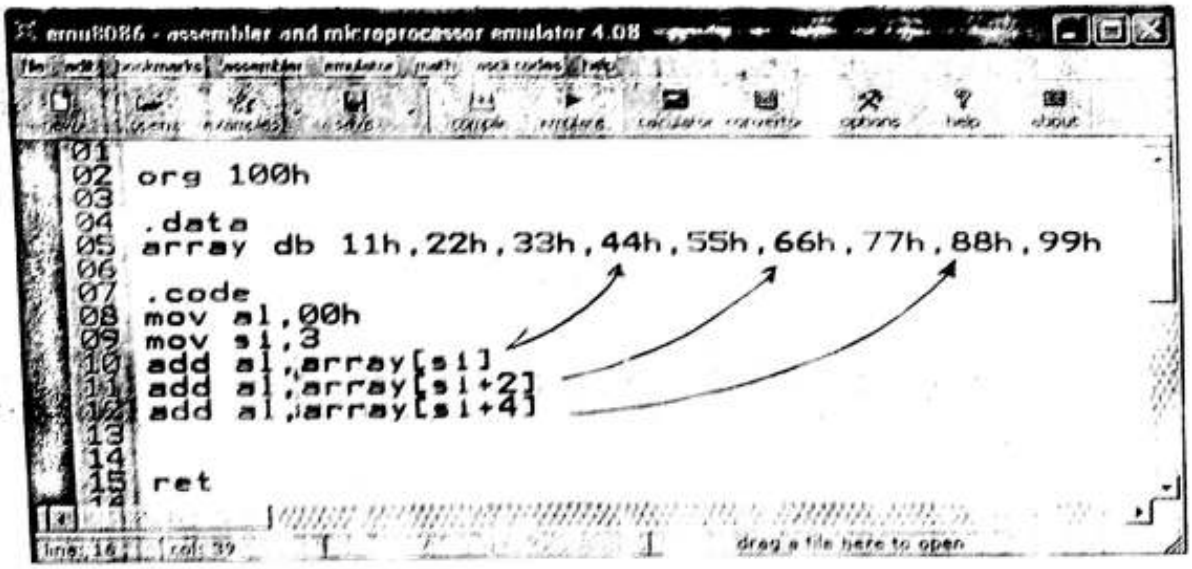
- **Example:**

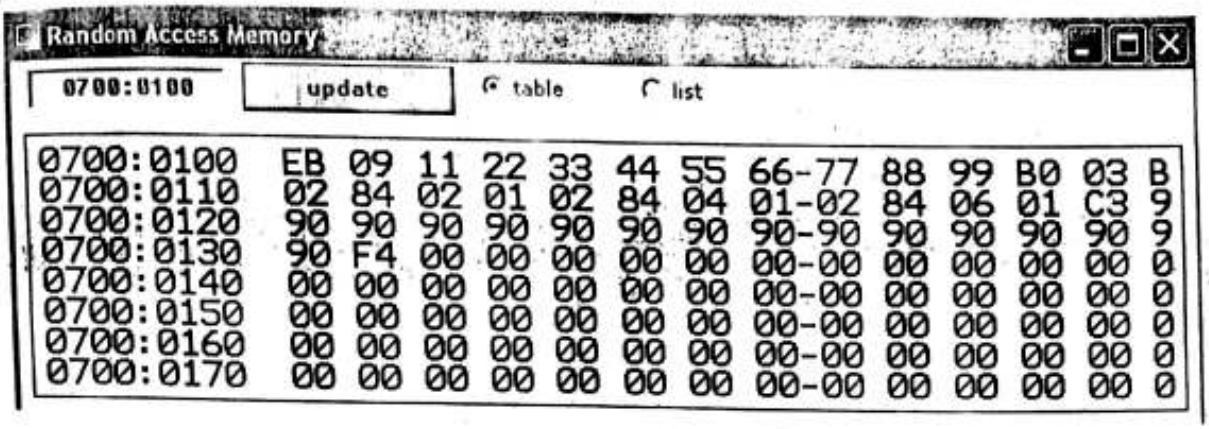
```
MOV DI, OFFSET NUMB
MOV AL, 0
ADD AL, [DI]
ADD AL, [DI+1]
```

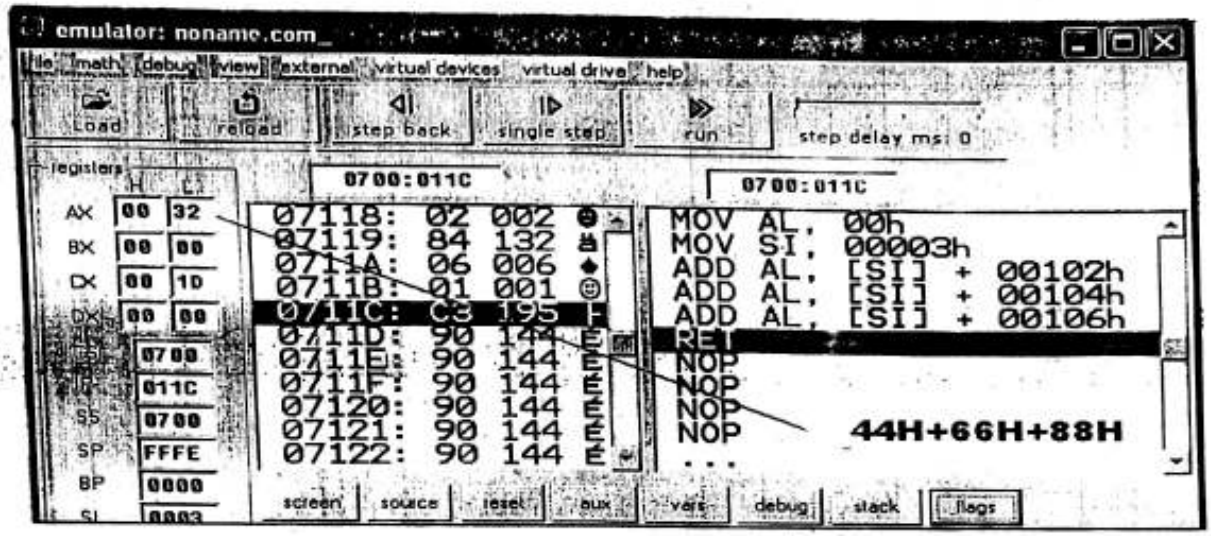
- **Array Addition**
- **Memory arrays are sequential lists of data.**

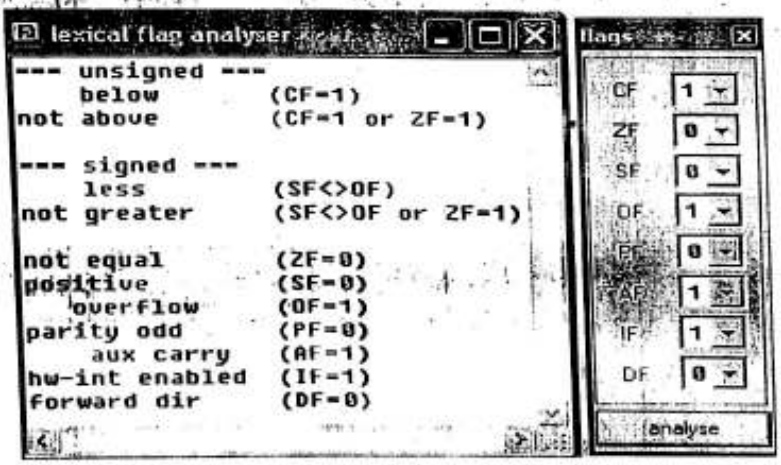
- **Example:**
- **Suppose we want to add elements 3, 5, and 7 of an area of memory called ARRAY.**

```
MOV AL, 0                ; clear sum (AL)
MOV SI, 3                ; address element 3
ADD AL, ARRAY[SI]        ; add element 3
ADD AL, ARRAY[SI+2]      ; add element 5
ADD AL, ARRAY[SI+4]      ; add element 7
```





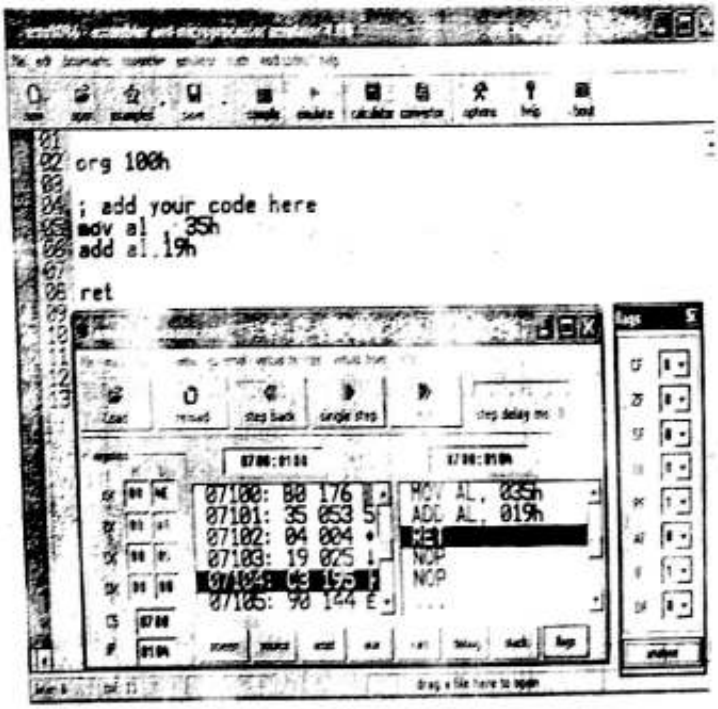




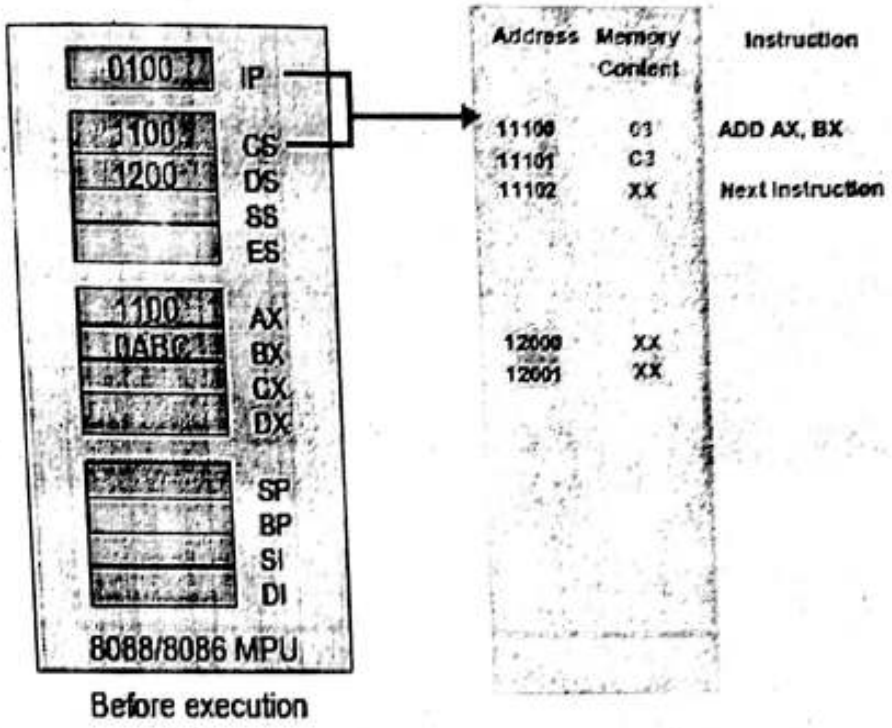
ADD AL,BL	AL = AL + BL
ADD CX,DI	CX = CX + DI
ADD BP, AX	BP = BP + AX
ADD CL,44H	CL = CL + 44H
ADD BX,245FH	BX = BX + 245FH
ADD [BX],AL	AL adds to the contents of the data segment memory location address by BX with the sum stored in the same memory location
ADD CL,[BP]	The byte contents of the stack segment memory location addressed by BP add to CL with the sum stored in CL
ADD BX,[SI + 2]	The word contents of the data segment memory location addressed by the sum of SI plus 2 add to BX with the sum stored in BX
ADD CL,TEMP	The byte contents of the data segment memory location TEMP add to CL with the sum stored in CL
ADD BX,TEMP[DI]	The word contents of the data segment memory location addressed by TEMP plus DI add to BX with the sum stored in BX

Carry and Overflow Summary

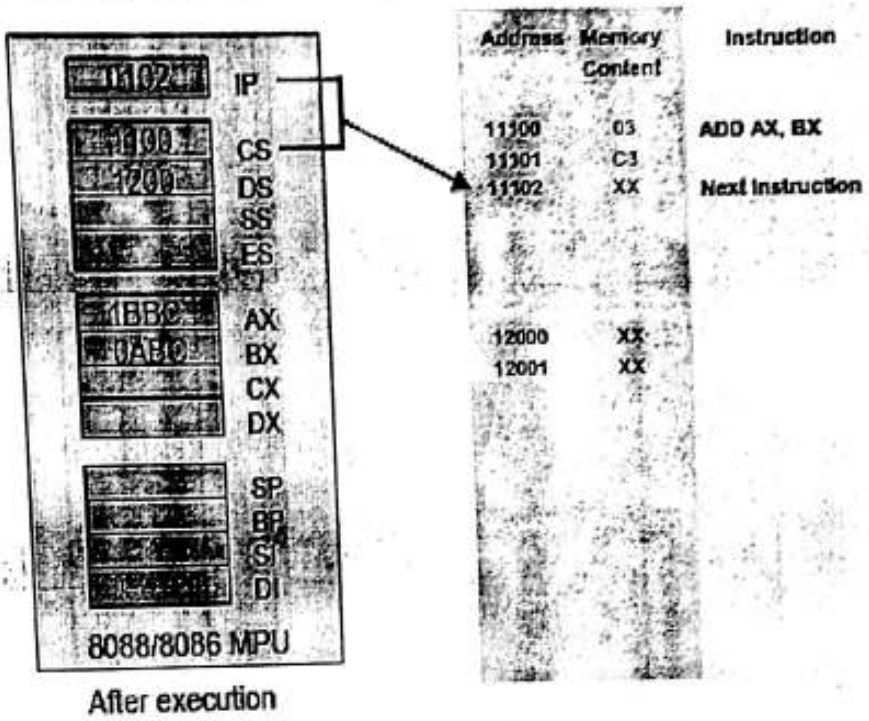
```
0000  B0 35   MOV  AL,35H
0002  04 19   ADD  AL,19H
              sum = 4EH in AL, C=0, O=0
0004  B0 35   MOV  AL,35H
0006  04 5B   ADD  AL,5BH
              sum = 90H in AL, C=0, O=1
0008  B0 35   MOV  AL,35H
000A  04 D3   ADD  AL,D3H
              sum = 08H in AL, C=1, O=0
000C  B0 9E   MOV  AL,9EH
000E  04 D3   ADD  AL,D3H
              sum = 71H in AL, C=1, O=1
```



ADD AX, BX



Before execution



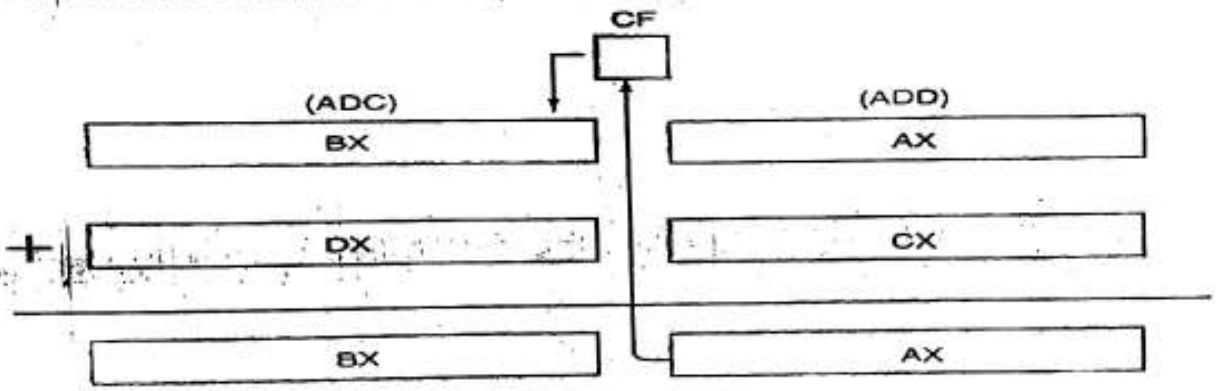
After execution

ADC Destination, Source

- **Destination + Source + Carry Flag → Destination**
- Destination and Source operands cannot be memory locations at the same time
- It modifies flags AF CF OF PF SF ZF
- An addition-with-carry instruction (ADC) adds the bit in the carry flag (C) to the operand data. This instruction mainly appears in software that adds numbers that are wider than 16 bits in the 8086.

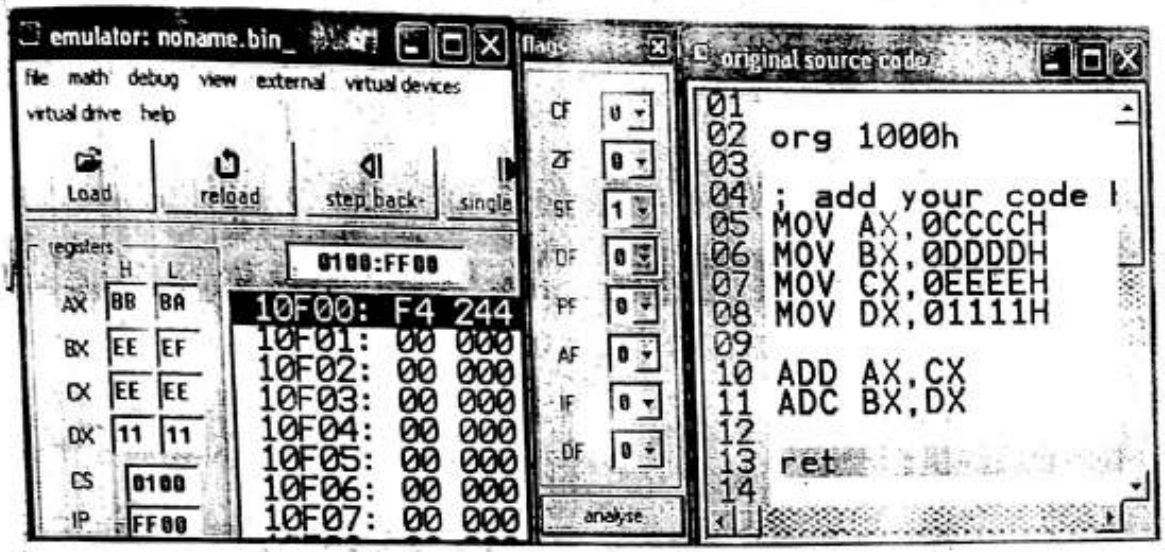
Assembly Language	Operation
ADC AL,AH	AL = AL + AH + carry
ADC CX,BX	CX = CX + BX + carry
ADC DH,[BX]	The byte contents of the data segment memory location addressed by BX add to DH with carry with the sum stored in DH
ADC BX,[BP + 2]	The word contents of the stack segment memory location address by BP plus 2 add to BX with carry with the sum stored in BX

- Suppose a program is written for the 886 to add the 32 bit number in BX and AX to the 32 bit number in DX and CX.



EXAMPLE

Perform a 32-bit binary add operation on the contents of the processor's register.



ADD [BX], DX

	Before	After
DX	1234H	
BX	1000H	
DS:1000H	2000H	3234H
DS:1002H		

In 3234H, 34H has three 1's. So P flag =0

ADC DH ,[SI]
Add with Carry

81H
1000 0001B(Two 1's)

	Before	After
DH	30H	81H
Carry flag	1	0
SI	2000H	
DS:2000H	50H	
DS:2001H	60H	

New flag values: Ac=0, S=1, Z=0, V=1, P=1

INC Destination

- **INC Src**
- **It increments the byte or word by one.**
- **The INC instruction adds 1 to any register o memory location, except a segment register.**
- **The operand can be a register or memory location.**
- **It effects AF, OF, PF, SF, ZF flags.**
- **CF is not effected.**

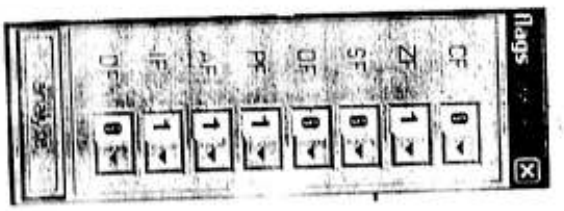
Destination
Reg16
Reg8
Memory

Allowed operands for INC instruction

Note:-
Carry doesn't change because we often use increment in programs that depend upon the contents of the carry flag.

Assembly Language	*Operation*
INC BL	BL = BL + 1
INC SP	SP = SP + 1
INC BYTE PTR [BX]	Adds 1 to the byte contents of the data segment memory location addressed by BX
INC WORD PTR [SI]	Adds 1 to the word contents of the data segment memory location addressed by SI
INC DWORD PTR [ECX]	Adds 1 to the doubleword contents of the data segment memory location addressed by ECX
INC DATA1	Increments the contents of data segment memory location DAAT1

```
org 100h

mov Ax, 0FFFFh
INC AX

ret
```



AX = 0000
↑
ZF = 1

Binary Subtraction

- **Can subtract immediate data from a register or memory.**
- **Can subtract a register from a register.**
- **Can subtract a register from memory.**
- **Can subtract memory from a register.**
- **Can NOT subtract data in one memory location from that in another memory location.**

Recall Full Subtractor Truth Table

C_i	A_i	B_i	D_i	C_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

					Hex
C	1	1	0	0	
A	0	1	0	1	5
B	0	1	1	1	- 7
	1	1	1	0	E

Final borrow = 1

Deo	Hex
181	B5
- 111	- 6F
70	46

Binary

```
  1 0 0 1 1 1 0
  1 0 1 1 0 1 0 1
  0 1 1 0 1 1 1 1
  ---------------
  0 1 0 0 0 1 1 0
```

Final borrow = 0

Subtraction Instructions	
Mnemonic	Meaning
SUB ac, data	Subtract immediate data from AL or AX register
SUB mem/reg, data	Subtract immediate data from register or memory location
SUB mem/reg1, mem/reg2	Subtract register from register, register from memory, or memory from register
SBB ac, data	Subtract with borrow immediate data from AL or AX register
SBB mem/reg, data	Subtract with borrow immediate data from register or memory location
SBB mem/reg1, mem/reg2	Subtract with borrow reg. from reg., reg. from memory, or memory from reg.

- **SUB Des, Src**
- It subtracts a byte to byte or a word to word.
- It effects AF, CF, OF, PF, SF, ZF flags.
- For subtraction, CF acts as borrow flag.

- Example:

```
SUB AL, 74H   ; sub 74H from AL register
SUB DX, AX    ; sub AX from DX register
SUB AX, [BX]  ; sub [BX] from AX register
```

- **Register Subtraction**
- Subtracts the content of several registers.
- When arithmetic instructions executed, contents of the flag register change.
- Any SUB instruction modifies the contents of the sign, zero, carry, auxiliary carry, parity, and overflow flags.
- Example:

```
SUB AX, BX    ; sub BX from AX register
SUB AX, CX    ; sub CX from AX register
SUB AX, DX    ; sub DX from AX register
```

- **Memory-to-Register Subtraction**
- Moves memory data to be subtracted to a register.

- Example:

```
MOV DI, OFFSET NUMB
MOV AL, 0
SUB AL, [DI]
SUB AL, [DI+1]
```

❖ Immediate Subtraction

- ❖ **Immediate subtraction is employed whenever constant or known data are subtracted.**
- ❖ **Example:**

```
MOV CH, 22H
SUB CH, 44H
```

- ❖ **The subtraction is stored in CH register.**
- ❖ **Flags changes, as follows:**
- ❖ **Z = 0 (result not zero), S = 1 (result negative), C = 1 (carry), P = 1 (even parity), AC = 1 (half carry), O = 0 (no overflow).**

Assembly Language	*Operation*
SUB CL,BL	CL = CL – BL
SUB AX,SP	AX = AX – SP
SUB DH,6FH	H = DH – 6FH
SUB AX,0CCCCH	AX = AX – CCCCH
SUB [DI],CH	Subtracts the contents of CH from the contents of the data segment memory location addressed by DI
SUB CH,[BP]	Subtracts the byte contents of the stack segment memory location address by BP from CH
SUB AH,TEMP	Subtracts the byte contents of the data segment memory location TEMP from AH

SUBTRACTION OF UNSIGNED NUMBERS

SUB dest,source **; dest = dest - source**

- In subtraction 2's complement method is used.
- Execution of SUB instruction
  1. Take the 2's complement of the subtrahend (source operand)
  2. Add it to the minuend (destination operand)
  3. Invert the carry

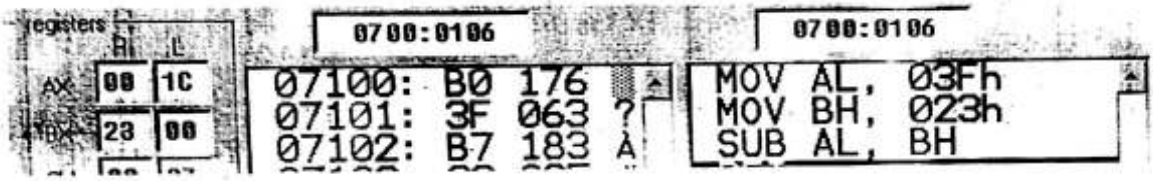
```
Ex:  MO  AL,3FH      ; load AL=3FH
     MO  BH,23H      ; load BH=23H
     SUB AL,BH       ; subtract BH from AL. Place result in AL
```

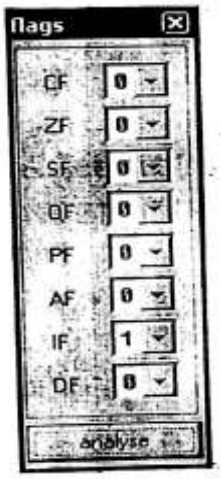
Execution steps:

AL	3F	0011 1111	0011 1111	
– BH	– 23	– 0010 0011	+ 1011 1101	(2's complement)
	1C	0001 1100	1 0001 1100	(CF=0) Step 3

CF=0, ZF=0, PF=0, SF=0.

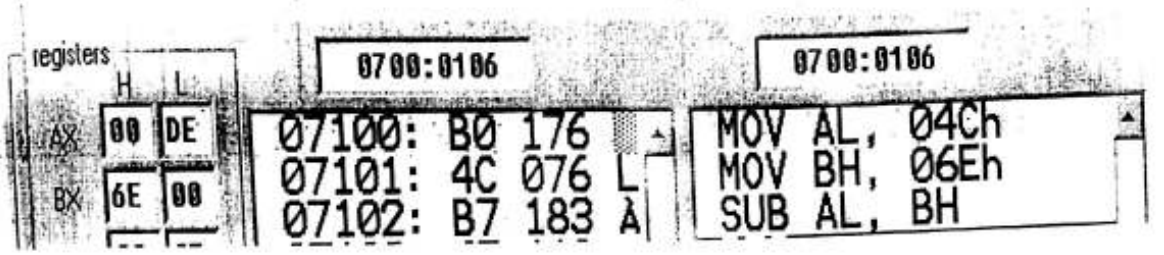
- If the CF=0, the result is positive and the destination has the result.
- If the CF=1, the result is negative and the destination has the 2's complement of the result. NOT and INC increment instructions can be used to change it.

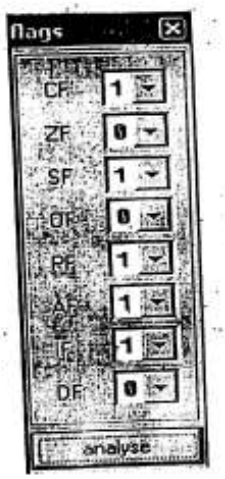




Analysis: Following the 3 steps for

```
4C  0100  1100                    0100 1100
-6E  0110  1110  2's comp       + 1001 0010
-22                               0 1101 1110   CF=1(Step 3) the result is
```



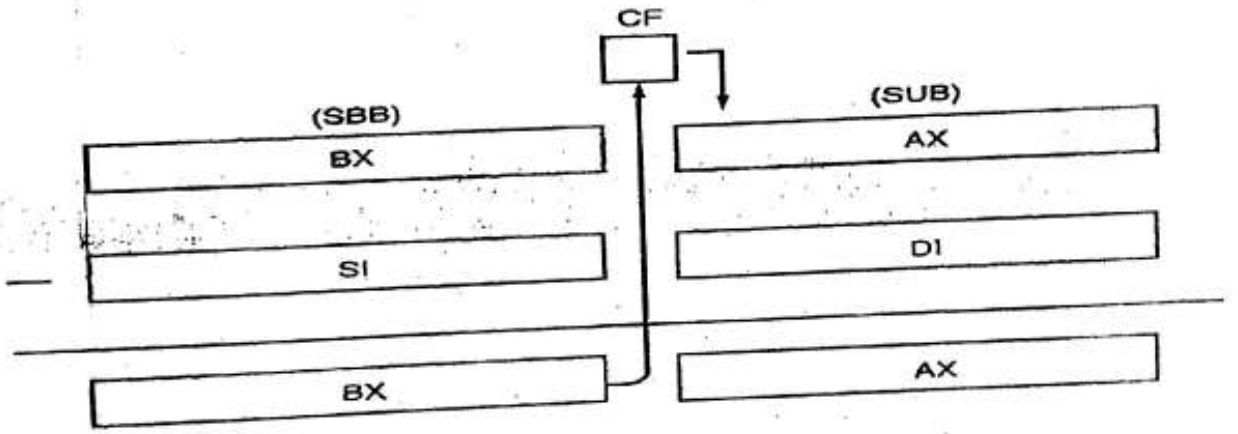


SBB Destination, Source

- Destination - Source - Carry Flag → Destination
- Used in multibyte (multiword) numbers.
- If CF=0, SBB works exactly like SUB
- If CF=1, SBB subtracts 1 from the result
-Destination and Source operands can not be memory locations at the same time
-It modifies flags AF CF OF PF SF ZF

Assembly Language	*Operation*
SBB AH,AL	AH = AH – AL – carry
SBB AX,BX	AX = AX – BX – carry
SBB CL,2	CL = CL – 2 – carry
SBB BYTE PTR[DI],3	Both a 3 and carry subtract from the contents of the data segment memory location addressed by DI
SBB [DI],AL	Both AL and carry subtract from the data segment memory location addressed by DI
SBB DI,[BP + 2]	Both carry and the word contents of the stack segment memory location addressed by the sum of BP and 2 subtract from DI

- The following example use the SUB instruction to subtract DI from AX, then uses SBB to subtract with borrow SI from BX.



SUB DH, CL
Subtract (without borrow)

	Before	After
DH	30H	0BH
CL	25H	

0BH =
0000 1011B(Three 1's)

New flag values: Ac=1, S=0, Z=0, V=0, P=0, Cy=0

SBB DH, CL
Subtract (with borrow)

	Before	After
DH	20H	FAH
Cy flag	1	1
CL	25H	

FAH =1111 1010(Six 1's)
2's complement of FAH=0000 0110 = +06 So, FAH = -06

New flag values: Ac=1, S=1, Z=0, V=0, P=1, Cy=1

Ex: Analyze the following program:

```
DATA_A      DD    62562FAH
DATA_B      DD    412963BH
RESULT      DD    ?
....        .....
            MOV   AX,WORD PTR DATA_A        ;AX=62FA
            SUB   AX,WORD PTR DATA_B        ;AX=AX – 963B
            MOV   WORD PTR RESULT,AX        ;save the result
            MOV   AX,WORD PTR DATA_A +2     ;AX=0625
            SBB   AX,WORD PTR DATA_B +2     ;SUB 0412 with borrow
            MOV   WORD PTR RESULT +2,AX     ;save the result
```

Note: PTR (Pointer) Directive is used to specify the size of the operand. Among the options for size are BYTE, WORD, DWORD and QWORD.

Solution:
After the SUB, AX = 62FA – 963B = CCBF and the carry flag is set. Since CF=1, when SBB is executed, AX = 625 – 412 – 1 = 212. Therefore, the value stored in RESULT is 0212CCBF.

Subtract 6FH from B5H

```
0000  B0 B5   MOV  AL,B5H
0002  2C 6F   SUB  AL,6FH

difference = 46H in AL
```

Dec	Hex	Binary
		1 0 0 1 1 1 0
-75	B5	1 0 1 1 0 1 0 1
-111	-6F	0 1 1 0 1 1 1 1
-186	46	C = 0 0 1 0 0 0 1 1 0
		O = 1

Thinking SIGNED we subtracted a positive number from a negative number and got a positive answer. This must be wrong. Therefore, the OVERFLOW bit, O, is set to 1.
Correct answer (-186) is outside the range -128 to +127.

Dec	Hex	Binary
		1 0 0 1 1 1 0
181	B5	1 0 1 1 0 1 0 1
-111	-6F	0 1 1 0 1 1 1 1
70	46	C = 0 0 1 0 0 0 1 1 0

This is the correct answer if we consider B5H to be UNSIGNED and equal to 181.
But suppose you were thinking of B5H = 10110101 as the 2's complement negative number 01001011 = 4BH or -75.

Dec	Hex		
			0 1 1 0 0 0 1
-75	B5	B5	1 0 1 1 0 1 0 1
-111	-6F	+91	1 0 0 1 0 0 0 1
-186	46	146	0 1 0 0 0 1 1 0

Ignore carry. Borrow = Carry, C = 0

Note no carry from bit 6 to bit 7 but there is a carry from bit 7 to C. Therefore, overflow, O = 1.

Take the two's complement of 6F and add
6FH = 01101111
10010001 = 91H

16-Bit Addition

```
  37FAH
+82C4H
 BABEH
```

```
0000  B8 FA 37   MOV  AX,37FAH
0003  05 C4 82   ADD  AX,82C4H

sum = BABEH in AX
```

16-Bit Subtraction

```
  A1C9H
-8315H
 1EB4H
```

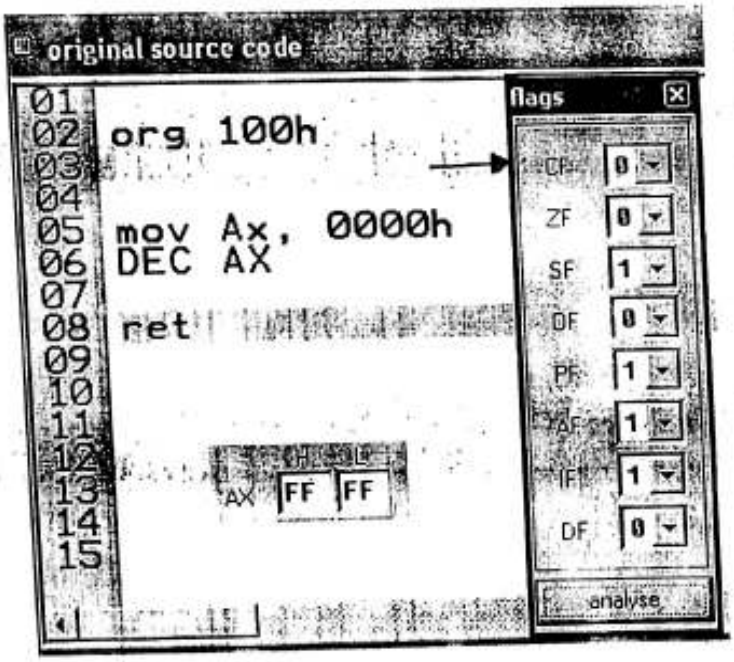
```
0000  B8 C9 A1   MOV  AX,A1C9H
0003  2D 15 83   SUB  AX,8315H

difference = 1EB4H in AX
```

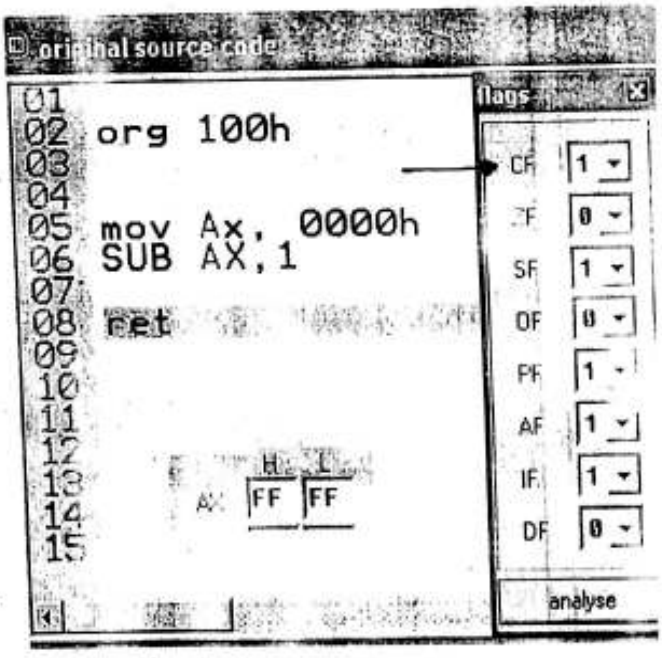
DEC Src

- It decrements the byte or word by one.
- The DEC instruction subtract 1 from any register or memory location, except a segment register.
- The operand can be a register or memory location.
- It effects AF, OF, PF, SF, ZF flags.
- CF is not effected.
- Example:

```
DEC AX      ; sub 1 from AX register
```

Assembly Language	Operation
DEC BH	BH = BH – 1
DEC CX	CX = CX – 1
DEC BYTE PTR [DI]	Subtracts 1 from the byte contents of the data segment memory location addressed by DI
DEC WORD PTR[BP]	Subtracts 1 from the word contents of the stack segment memory location addressed by BP
DEC NUMB	Subtracts 1 from the contents of the data segment memory location NUMB





NEG Destination

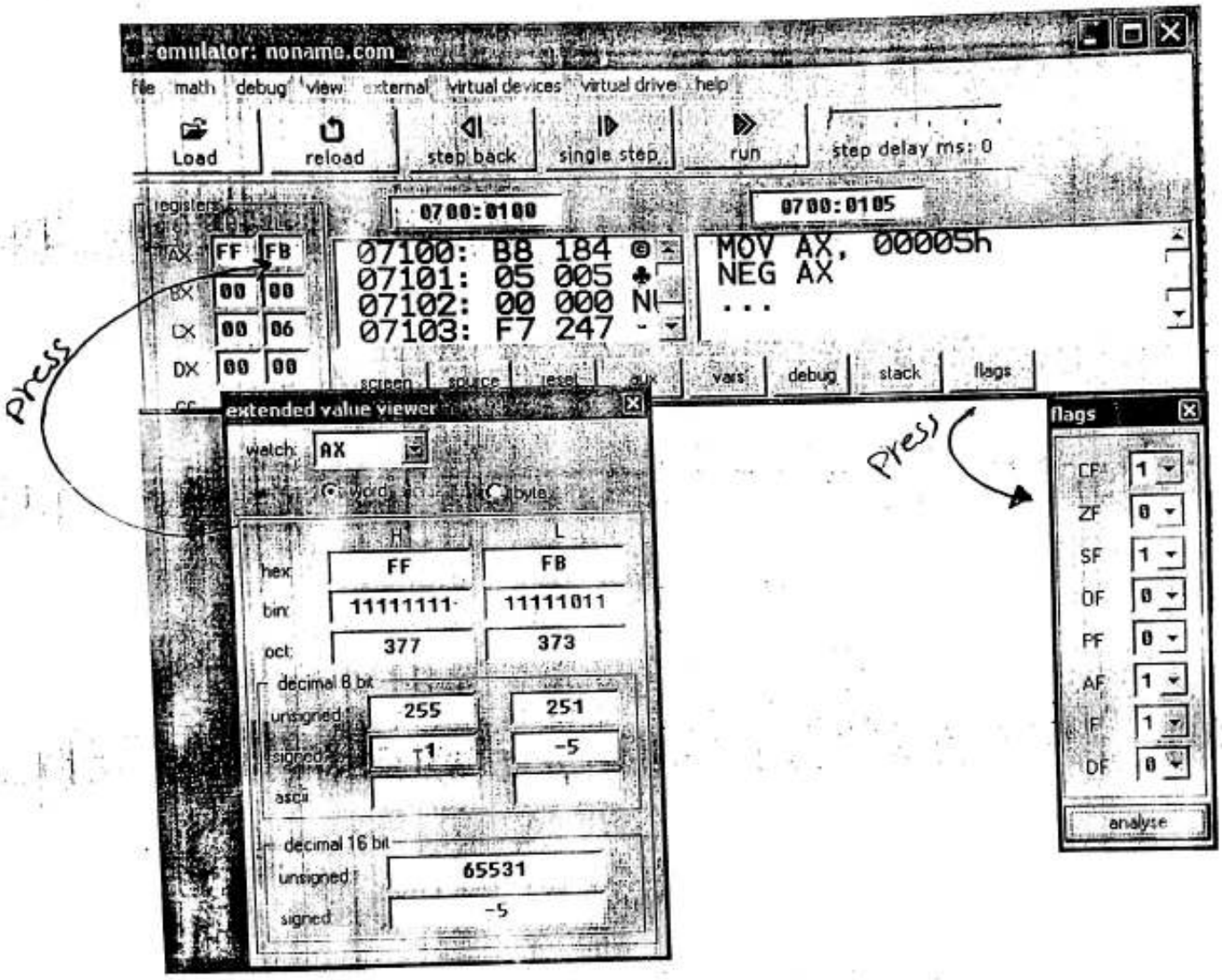
- Destination →0- Destination (the result is represented in 2's complement)
- Destination can be a register or a memory location
- It modifies flags AF CF OF PF SF ZF

Algorithm:

Invert all bits of the operand
Add 1 to inverted operand

Example:

```
MOV AL, 5  ; AL = 05h          AL = 0000 0101   05H
NEG AL     ; AL = 0FBh (-5)    AL = 1111 1011   FBH
NEG AL     ; AL = 05h (5)
```



Comparison Instruction:

Format: CMP OPERAND1 , OPERAND2
Operation: (OPERAND1) - (OPERAND2) affects the status flags.

	OPERAND1	OPERAND2
1	Reg	Reg
2	Reg	Mem
3	Mem	Reg
4	Reg	Imm
5	Mem	Imm
6	Acc	Imm

- **CMP Des, Src**
- **It compares two specified bytes or words.**
- **The Src and Des can be a constant, register or memory location.**
- **Both operands cannot be a memory location at the same time.** & segment register
- **The comparison is done simply by internally subtracting the source from destination.**
- **The value of source and destination does not change, but the flags are modified to indicate the result.**
- **The comparison instruction (CMP) is a subtraction that changes only the flag bits.**
- **Useful for checking the contents of a register or a memory location against another value.**
- **A CMP is normally followed by a conditional jump instruction, which tests the condition of the flag bits.**
- **Example:**

```
CMP AL, 10H
JAE NEXT        ; jump if above or equal
```

Compare operands	CF	ZF
Destination >source	0	0
Destination = source	0	1
Destination < source	1	0

Flag settings of the CMP instruction.

Assembly Language	*Operation*
CMP CL,BL	CL – BL
CMP AX,SP	AX – SP
CMP AX,2000H	AX – 2000H
CMP [DI],CH	CH subtracts from the contents of the data segment memory location addressed by DI
CMP CL,[BP]	The byte contents of the stack segment memory location addressed by BP subtract from CL
CMP AH,TEMP	The byte contents of the data segment memory location TEMP subtract from AH
CMP DI,TEMP[BX]	The word contents of the data segment memory location addressed by the sum of TEMP plus BX subtract from DI

CMP BH, CL

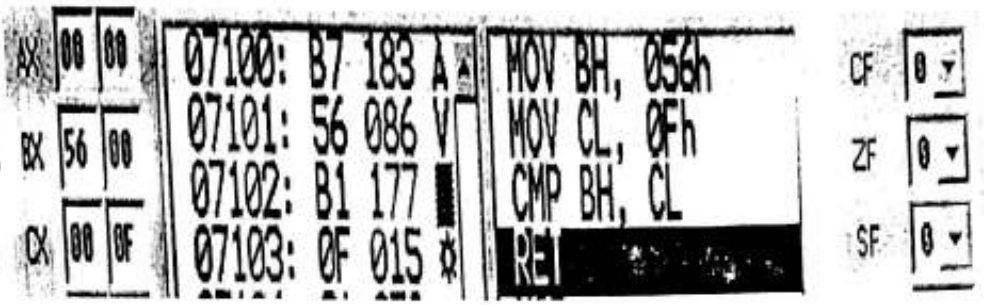
56H=01010110B

0FH=00001111B

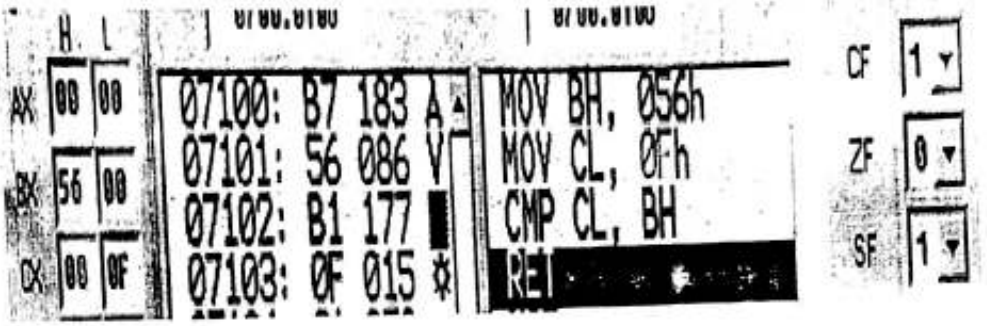
Only flags are affected

	Before	After
BH	56H	56H
CL	0FH	

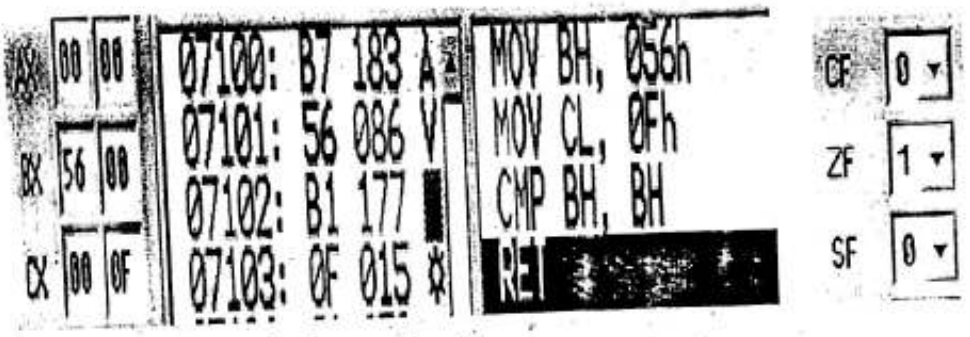
CMP BH, CL



CMP CL, BH



CMP BH, BH



BCD(Binary Coded Decimal
&
ASCII (American Standard Code for Information Interchange)
Instructions

Packed BCD Adjust Instructions

DAA	Decimal Adjust for Addition	Sum in AL adjusted to packed BCD format
DAS	Decimal Adjust for Subtraction	Difference in AL adjusted to packed BCD format

Unpacked BCD Arithmetic Instructions

AAA	Unpacked BCD Adjust for Addition	«AL» ← sum in AL adjusted to unpacked BCD format «AH» ← <AH> + carry from adjustment
AAS	Unpacked BCD Adjust for Subtraction	«AL» ← difference in AL adjusted to unpacked BCD format «AH» ← «AH» - borrow from adjustment
AAM	Unpacked BCD Adjust for Multiplication	«AX» ← product in AL adjusted to unpacked BCD format with AH containing high order digit
AAD	Unpacked BCD Adjust for Division	«AL» ← 10 * «AH» + «AL» «AH» ← 0

Binary representation of 0 to 9 (used by human beings) is called **BCD**.

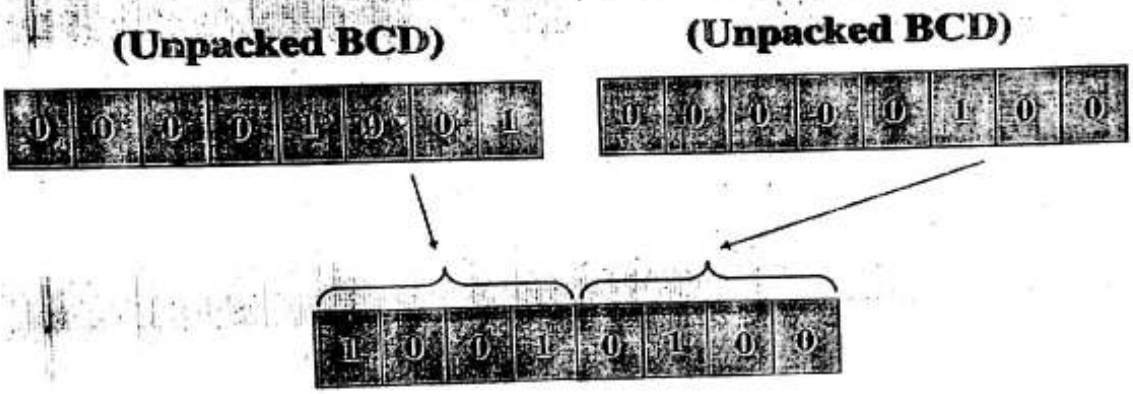
- There are two types of BCD numbers:
  1- Unpacked BCD **2- Packed BCD**

- **Unpacked BCD**: 1 byte is used to store 4 bit BCD code. E.g. **0000 1001** is unpacked BCD for **9**.

- **Packed BCD**: 1 byte is used to store two 4 bit BCD codes. E.g. **0101 1001** is packed BCD for **59**. More efficient in storing data.

Digit	BCD
0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
4	0000 0100
5	0000 0101
6	0000 0110
7	0000 0111
8	0000 1000
9	0000 1001

Use 8 bits to store 2 BCD digits



(Packed BCD)

Digit	ASCII
0	(30H) 011 0000
1	(31H) 011 0001
2	(32H) 011 0010
3	(33H) 011 0011
4	(34H) 011 0100
5	(35H) 011 0101
6	(36H) 011 0110
7	(37H) 011 0111
8	(38H) 011 1000
9	(39H) 011 1001

- **7-bit representation.**
- **Keyboards, printers, and monitors are all in ASCII.**
- **To process data in BCD, ASCII data should be converted first to BCD.**
- **Remember that ASCII code representation of a number is : number + 30 H**
- **ASCII Code for (3) is : 3 +30 = $(33)_{ASCII}$**

ASCII numbers

Key	ASCII(Hex)	Binary	BCD (Unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

Packed BCD	Unpacked BCD	ASCII
29H	02 & 09	32 & 39
0010 1001	0000 0010 & 0000 1001	0011 0010 & 0011 1001

BCD Arithmetic

Two arithmetic techniques operate with BCD data: **addition** and **subtraction**.

1- DAA (decimal adjust after addition)

- instruction follows BCD addition .
- It is used to make sure that the result of adding two BCD numbers is adjusted to be a correct BCD number.
- It only works on AL register.

2- DAS (decimal adjust after subtraction)

- instruction follows BCD subtraction.
- It is used to make sure that the result of subtracting two BCD numbers is adjusted to be a correct BCD number.

It only works on AL register.

Both DAA and DAS correct the result of addition or subtraction so it is a BCD number.

Packed BCD Adjust Instructions

DAA	Decimal Adjust for Addition	Sum in AL adjusted to packed BCD format
DAS	Decimal Adjust for Subtraction	Difference in AL adjusted to packed BCD format

Ex1:

```
MOV AL, 17H
ADD AL, 28H
```

Result=**3FH**
(Not a BCD)

Ex2:

```
MOV AL,52H
ADD AL,87H
```

Result=**D9H**
(Not a BCD)

• To solve these problems add 6 to the lower nibble of 3FH and upper nibble of D9H.

3F + 6 = 45H **D9 + 60 = 139H**

• Now the results are BCD.
• There is a special instruction to do this correction: **DAA**

DAA: Decimal Adjust Accumulator

This instruction is used to convert the result of the addition of two packed BCD numbers to a valid BCD number. The result has to be only in AL.

DAA	No operands	Decimal adjust After Addition.
		Corrects the result of addition of two packed BCD values.
		Algorithm:
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL + 6
		• AF = 1
		if AL > 9Fh or CF = 1 then:
		• AL = AL + 60h
		• CF = 1

Summary of DAA action

1. If after an ADD or ADC instruction the lower nibble (4 bits) is greater than 9, or if AF=1, add 0110 to the lower 4 bits.
2. If the upper nibble is greater than 9, or CF =1, add 0110 to the upper nibble.
3. DAA works only after the ADD and ADC instruction. (E.g. it doesn't work with INC instruction) ,
4. In addition the destination operand must be AL in order for DAA to work.
5. Note that in BCD addition the operands can never have any digit greater than 9.

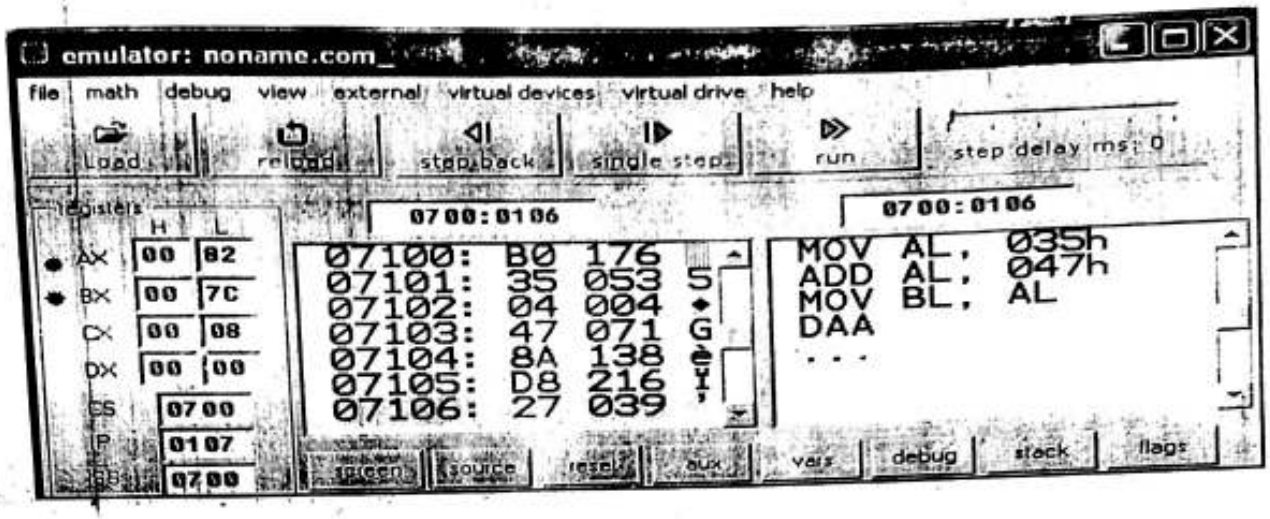
Note: AF (Auxiliary carry Flag) is only used for BCD addition and correction.

C	Z	S	O	P	A
↙	↙	↙	↙	↙	↙

BCD Addition

Binary		Decimal (BCD)	
35H	00110101	35H	00110101
+47H	01000111	+47H	01000111
7CH	01111100	82H	10000010

```
0000  B0 35   MOV  AL, 35H   ;AL = 35H
0002  04 47   ADD  AL, 47H   ;AL = AL+47H
0004  27      DAA            ;Decimal adjust
```



Ex:

Hex	BCD	
29	0010 1001	
+18	+0001 1000	
41	0100 0001	AF=1
+6	+ 0110	Because AF=1 DAA will add 6 to the lower nibble
47	0100 0111	The final result is BCD

Eg. AL = 53 H CL = 29 H

```
ADD AL, CL      ; AL  (AL) + (CL)
                ; AL  53 + 29
                ; AL  7C
DAA             ; AL  7C + 06 (as C>9)
                ; AL  82
```

DAS : Decimal Adjust after Subtraction

- This instruction converts the result of the subtraction of two packed BCD numbers to a valid BCD number. The subtraction has to be in AL only.
- The problem associated with the addition of packed BCD numbers also shows up in subtraction.
- DAS is used to correct this problem.
- DAS must come after SUB or SBB instructions.
- AL must be used as the destination register in subtraction for the DAS to work

Eg. AL = 75h BH = 46h

```
SUB AL, BH      ; AL  2F = (AL) - (BH) = 2Fh
                ; AF = 1
DAS             ; AL  29  (as F>9, F - 6 = 9)
```

DAS	No operands	Decimal adjust After Subtraction.
		Corrects the result of subtraction of two packed BCD values.
		Algorithm:
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL - 6
		• AF = 1
		if AL > 9Fh or CF = 1 then:
		• AL = AL - 60h
		• CF = 1

Summary of DAS action

1. If after an SUB or SBB instruction the lower nibble is greater than 9, or if AF=1, subtract 0110 from the lower 4 bits.

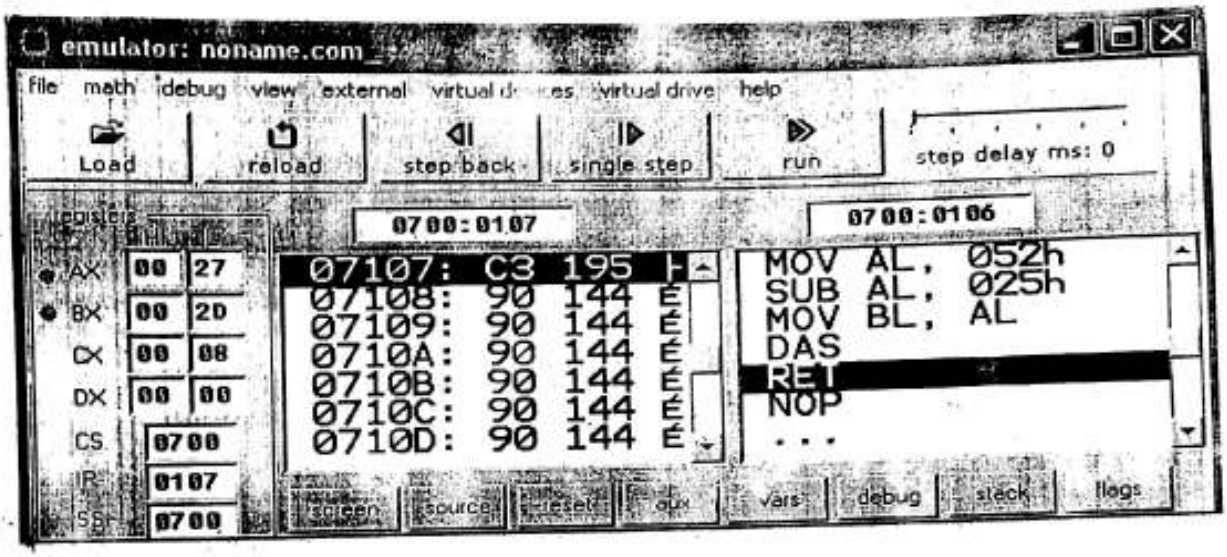
2. If the upper nibble is greater than 9, or CF =1, subtract 0110 from the upper nibble.

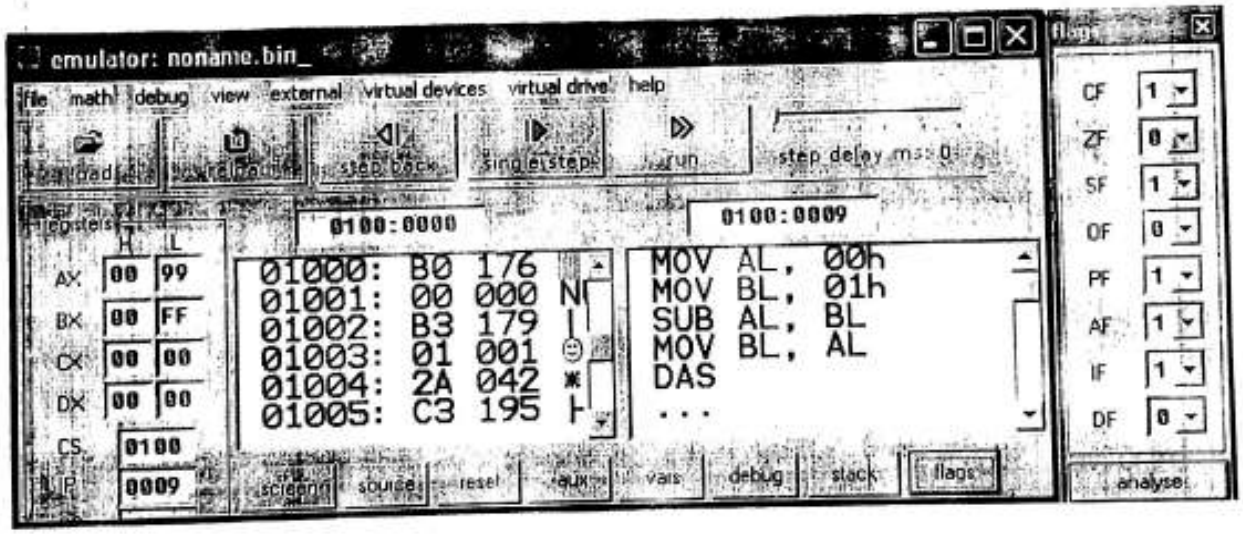
C	Z	S	O	P	A
←	←	←	←	←	←

BCD Subtraction

Binary		Decimal (BCD)	
52H	01010010	52H	01010010
-25H	00100101	-25H	00100101
2DH	00101101	27H	00100111

```
0000    B0 52    MOV   AL,52H    ;AL = 52H
0002    2C 25    SUB   AL,25H    ;AL = AL-25H
0004    2F       DAS              ;Decimal adjust
```





ASCII Arithmetic

- **ASCII arithmetic instructions function with coded numbers, value 30H to 39H for 0–9.**
- **Four instructions in ASCII arithmetic operations:**
  - **AAA (ASCII adjust after addition)**
  - **AAD (ASCII adjust before division)**
  - **AAM (ASCII adjust after multiplication)**
  - **AAS (ASCII adjust after subtraction)**

Unpacked BCD Arithmetic Instructions

AAA	Unpacked BCD Adjust for Addition	«AL» ← sum in AL adjusted to unpacked BCD format «AH» ← <AH> + carry from adjustment
AAS	Unpacked BCD Adjust for Subtraction	«AL» ← difference in AL adjusted to unpacked BCD format «AH» ← «AH» - borrow from adjustment
AAM	Unpacked BCD Adjust for Multiplication	«AX» ← product in AL adjusted to unpacked BCD format with AH containing high order digit
AAD	Unpacked BCD Adjust for Division	«AL» ← 10 * «AH» + «AL» «AH» ← 0

AAA : ASCII Adjust After Addition

The AAA instruction is executed after an ADD instruction that adds two ASCII coded operand to give a byte of result in AL. ***The AAA instruction converts the resulting contents of Al to an unpacked decimal digits.***

AAA	No operands	ASCII Adjust after Addition.
		Corrects result in AH and AL after addition when working with BCD values.
		It works according to the following Algorithm:
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL + 6
		• AH = AH + 1
		• AF = 1
		• CF = 1
		else
		• AF = 0
		• CF = 0
		In both cases:
		clear the high nibble of AL.

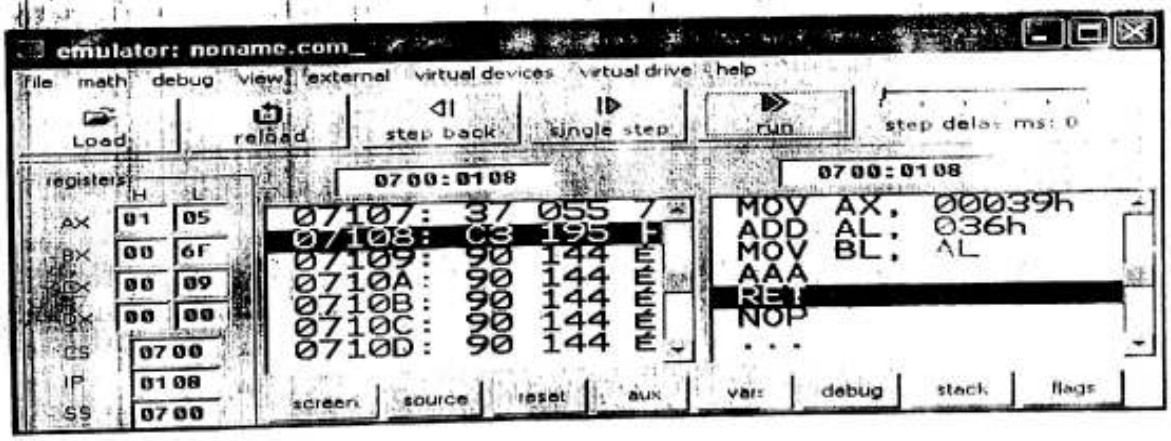
C	Z	S	O	P	A
↵					↵

```
Eg.  ADD CL, DL    ;  [CL] = 32H = ASCII for 2
                   ;  [DL] = 35H = ASCII for 5
                   ;  Result [CL] = 67H
     MOV AL, CL    ;  Move ASCII result into AL since
                   ;  AAA adjust only [AL]
     AAA           ;  [AL]=07, unpacked BCD for 7
```

ASCII Addition

Decimal		ASCII	
9		39	
+6		+36	
15		01	05
		AH	AL

```
0000   B8 39 00   MOV   AX,0039H    ;AL = 39H
0003   04 36      ADD   AL,36H      ;AL = AL + 36H
0005   37         AAA               ;ASCII adjust
```



AAS : ASCII Adjust AL after Subtraction

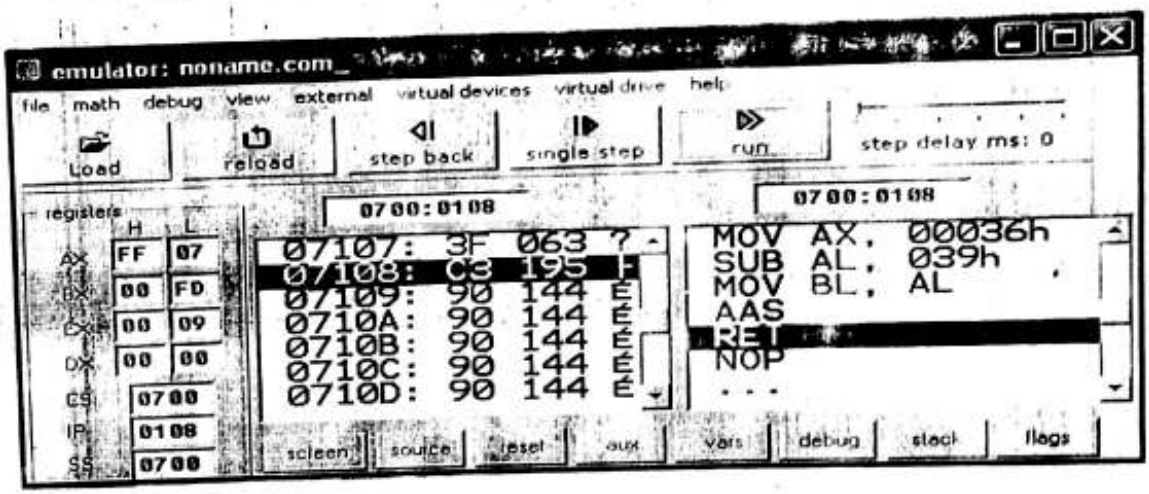
This instruction corrects the result in AL register after subtracting two unpacked ASCII operands. The result is in unpacked decimal format. The procedure is similar to AAA instruction except for the subtraction of 06 from AL.

AAS	No operands	ASCII Adjust after Subtraction.
		Corrects result in AH and AL after subtraction when working with BCD values.
		Algorithm:
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL - 6
		• AH = AH - 1
		• AF = 1
		• CF = 1
		else
		• AF = 0
		• CF = 0
		In both cases:
		clear the high nibble of

ASCII Subtraction

Decimal	ASCII
6	36
-9	-39
Borrow = 1 7	FF 07 (Carry flag = 1)
	AH AL

```
0000   B0 36 00   MOV  AX,0036H   ;AL = 36H
0003   2C 39       SUB  AL,39H    ;AL = AL - 39H
0005   3F         AAS            ;ASCII adjust
```



```
Ex:  MOV  AL,'5'      ;AL=35
     ADD  AL,'2'      ;add to AL 32 the ASCII for 2
     AAA              ;change 67H to 07H
     OR   AL,30       ;OR AL with 30H to get ASCII

Ex:  SUB  AH,AH       ;AH=00
     MOV  AL,'7'      ;AL=37H
     MOV  BL,'5'      ;BL=35H
     ADD  AL,BL       ;37+35=6CH  therefore AL=6C
     AAA              ;changes 6CH to 02 in AL and AH=CF=1
     OR   AX,3030H    ;AX=3132 which is the ASCII for 12H
```

Note: AAA and AAS will only work on the AL register. The data can be unpacked BCD rather than ASCII.

```
Ex:  MOV  AX,0105H    ;AX=0105 unpacked BCD for 15
     MOV  CL,06       ;CL=06H
     SUB  AL,CL       ;5-6=-1 (FFH)
     AAS              ;FFH  in AL is adjusted to 09 and
                      ;AH is decremented leaving AX=0009
```

❖ **Example:**

❖ **38 – 39 = FF, the result should be FF 39H**

```
MOV AH, 00          ; AH = 00H
MOV AL,'8'          ; AX = 0038H
SUB AL, '9'         ; AX = 00FFH
AAS                 ; AX=FF09H
OR AX, 30H          ; AX = FF09
```

❖ **Example: Positive Result**

```
SUB AH, AH          ; AH = 00H
MOV AL,'9'          ; AL = 39H
SUB AL, '3'         ; AL =39H-33H = 06H
AAS                 ; AX=0006H
OR AL, 30H          ; AL = 36H
```

1. Write an ALP (assembly language programming) for addition of two 8-bit data BB H and 11 H.

```
MOV AL, BB H    : 8-bit data BB H into AL
MOV CL, 11 H    : 8-bit data 11 H into CL
ADD AL, CL      : Contents of AL and CL added
HLT             : Stop.
```

Comment : Result in AL = CC H.

2. Write an ALP for addition of two 16-bit data BB11 H and 1122 H.

```
MOV AX, BB11 H  : 16-bit data BB11 H into AX
MOV CX, 1122 H  : 16-bit data 1122 H into CX
ADD AX, CX      : Contents of AX and CX added
HLT             : Stop
```

Comment : Result in AX = CC33 H.

3. Write an ALP for addition of two 8-bit data BB H and 11 H. The first data has an offset address of 0304 H and displacement 07.

```
MOV BX, 0304 H      : Offset address put in BX
MOV AL, 11 H        : 8-bit data 11H into AL
ADD AL, [BX + 07]   : 8-bit data from offset + displacement added with AL
HLT                 : Stop.
```

Comment : Result in AL = CC H.

4. Write an ALP that subtracts 1234 H existing in DX from the word beginning at memory location MEMWDS.

```
MOV DX, 1234 H      : 16-bit data 1234 H put into DX
SUB MEMWDS, DX      : Subtract data word 1234 H existing in DX from the data word
                      pointed to by MEMWDS.
HLT                 : Stop.
```

Comment : If MEMWDS points to 3000 H then,
[3001 H : 3000 H] ← [3001 H : 3000 H] – 1234 H

5. Write an ALP which multiplies two 8-bit data 21 H and 17 H.

```
MOV AL, 21 H    : 8-bit multiplicand 21 H put into AL
MOV CL, 17 H    : 8-bit multiplier 17 H put into CL
MUL CL          : Contents of CL and AL are multiplied and the result
                  stored in AX
HLT : Stop.
```

Comment : Result in AX = 02F7 H.

6. Write an ALP for dividing 1234 H by 34 H.

```
MOV AX, 1234 H  : 16-bit dividend in 1234 H
MOV CL, 34 H    : 8-bit divisor in 34 H
DIV CL          : Content of AX divided by content of CL
HLT             : Stop.
```

Comment: Result in AX with Quotient in AL = 59 H and Remainder in AH = 20 H.

7. Write an ALP for ASCII addition of two numbers 2 H and 5 H.

```
MOV AL, 32 H    : ASCII code 32 H for number 2 H is moved into AL
MOV BL, 35 H    : ASCII code 35 H for number 5 H is moved into BL
AAA             : ASCII adjust for addition
HLT             : Stop.
```

Result : (AL) - 07 H.

8. Write an ALP to evaluate X (Y + Z), where X = 10 H, Y = 20 H and Z = 30 H.

```
MOV AL, 20 H    : 20 H put in AL
MOV CL, 30 H    : 30 H put in CL
ADD AL, CL      : AL and CL are added up and result in AL
MOV CL, AL      : AL transferred in CL
MOV AL, 10 H    : 10 H put in AL
MUL CL          : AL and CL are multiplied and result in AL
MOV SI, 4000 H  : Source address in SI
MOV SI, AL      : AL put in SI
HLT : Stop.
```

9- Write an ALP to evaluate $X^3 + 10$, X, is present in the data segment of memory at offset 2000h and store the result in 3000h.

```
MOV  AL , [2000h]
MOV  AH , 0h
MOV  BL , AL
MUL  AL
MUL  BL
ADD  AX ,000Ah
MOV [3000h] ,AX
```

10- Write an ALP that transfers a block of 100 bytes of data. The source and destination memory blocks start at 3000 H and 4000 H memory locations respectively. The data segment register value is 1000h.

```
    MOV AX, 1000h   : Move initial address of DS register into AX.
    MOV DS, AX      : DS loaded with AX
    MOV SI, 3000 H  : Source address put into SI.
    MOV DI, 4000 H  : Destination address put into DI.
    MOV CX, 64 H    : Count value for number of bytes put into CX register
xx: MOV AH, [SI]    : Source byte moved into AH
    MOV [DI], AH    : AH byte moved into destination address
    INC SI          : Increment source address
    INC DI          : Increment destination address
    DEC CX          : Decrement CX count
    JNZ xx          : Jump to 200D H until CX = 0
    HLT : Stop.
```

2- Find the values of the BX register, as you execute each line of the program.

```
0100:1000  AA BB CC DD EE FF 11 22
```

```
MOV bx,offset[1000h]
```
BX= _____

```
MOV bx,offset 1000h
```
BX= _____

```
MOV bx,1000
```
BX= _____

```
MOV bx,[1000h]
```
BX= _____

```
MOV bx,[1000000000100b]
```
BX= _____

[5]

Q3- Use the following debug screen:

```
AX=0BD9  BX=0000  CX=008D  DX=0000  SP=021C  BP=0000  SI=0000  DI=0000
DS=F000  ES=0001  SS=1000  CS=000F  IP=000A
000F:000A 8ED8          MOV     DS,AX
```

a. Content of AL register ? .....
b. Physical address of current instruction (MOV DS , AX) ? .....
c. Memory Size of current instruction (MOV DS , AX) in bytes ? .....
d. Logical Address of last byte in the Data Segment ? .....
e. What is address of next instruction? .....

[5]

Q4) What addressing mode is used the following instruction

a. MOV AX, BX .....
b. MOV AX, [DI] .....
c. MOV AX,[BP + SI] .....
d. MOV CX, 2342H .....
e- MOV AX, [BP+10h] .....

[5]

Q5) Assuming 16-bit Intel instructions translate from machine to assembly code (a) and translate from assembly code t machine (b). *Note*: if D=1, data flows to REG from R/M and, if D=0, data flows to R/M from REG field.

a. 8AF3 Assembly code .....

b. MOV [SI], AH Machine code .....

[5]

Q6) For given table solve questions

00000 H	15 H
00001 H	9A H
00002 H	47 H
00003 H	EA H
00004 H	C7 H
.	
.	
FFFFB H	0D H
FFFFC H	1A H
FFFFD H	BC H
FFFFE H	04 H
FFFFF H	B5 H

a- Find the LSB of byte-content stored in physical address (PA) of 00003 .....
b- Find the physical address for the word content of C7EA : .....
c- Find the most-significant-byte of the Word stored in PA of FFFFD .....
d- Find the 1st misaligned-words stored in the memory .....
e- d- Find the 1st aligned-words stored in the memory .....

Q7- If the contents of CPU registers are as follow: DS= 2003H, ES= 4043H, SS= 5000H, SI=1008H , DI=107AH , BX=100AH , BP= 2000 H, IP=1012, Find the following answers:

a- Find the physical address of the last storage location in Extra Segment of main memory

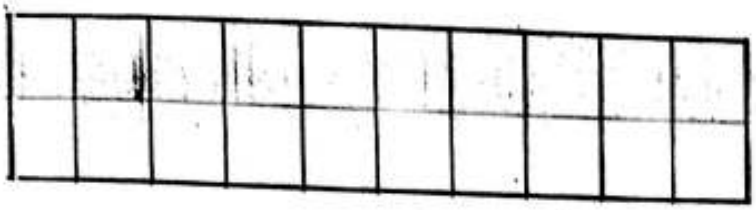
b- Using Register Indirect addressing mode, write a program to load DX register with the word contents of main memory location 210AAH.

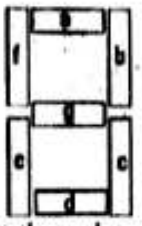
c- Using Based plus Indexed addressing mode, write a program to load AX register with the word contents of ma memory location 53008H.

الرقم ..... المجموعة ..... رقم القيد ..... الاسم .....

Q1A) Suppose that a seven segment LED display lookup table is stored in memory at address TABLE, use this table to translate the BCD number in AL to a seven segment code also in AL. Let TABLE =1000H, DS=1000H,

1- Obtain the look up table?





2- Write code to read the input key from port (1234H) and output the value to a 7-segemnt display at port 3000H.

b) Let num1=11223344H and num2=55667788H are stored at memory locations200 and 300 respectively in the current data segment. ADD num1and num2 and store the result at memory location 400.

```
MOV AX, [0200]
MOV BX, [0202]
ADD AX, [300]
ADC BX, [0302]
MOV [400], AX
MOV [402], BX
```

Q2A) What is the difference between :

1. near call & far call .....
2. JA & JNBE .....
3. LOOPNE & LOOPNZ .....
4. SHL AL,2 & SAL AL,2 .....

B) What is the result of executing the following instruction sequence?

```
MOV AX, 00075h
MOV BX, 00032h
ADD AX, BX
DAA
```

AX H L CF

```
MOV AX, 00037h
MOV BX, 00035h
ADD AX, BX
AAA
OR AX, 03030h
```

AX H L CF

```
MOV AL, 036h
MOV BL, 031h
IMUL BL
AAM
```

AX H L

```
MOV AX, 00405h
MOV BL, 0FCh
AAD
IDIV BL
```

AX H L

Q3A) 1- Which type of JMP instruction assembles if the distance is 0020 h bytes

a. near. b. far. c. short. d. none of the above.

2- Number of the times the instruction sequence below will loop before coming out of loop is

```
    MOV AL, 00h
A1: INC AL
    JNZ A1
```

a. 00 b. 01 c. 255 d. 256

3- The comparison instruction (CMP) is a subtraction that changes only the

a. Source b. destination c. flag bits. d. none of the above.

4. LOOPNE: Jump to specified label until

a. CX = 0 b. CX= 0 and CF=0 c. CX= 0 and ZF=0 d. CX= 0 and ZF=1

5. Intrasegment RET is

a. Near RET b. Far RET c. Intersegment RET d. none of the above.

B) Write ALP to find number of times letter 'E' exist in the string Electrical and Electronic Engineering'. Store the count at memory?

Q4A) Explain the instruction LEA, LDS and LES.

Mnemonic	Meaning	Format	Operation	Flags affected
LEA				
LDS				
LES				

B) Write a program to exchange a block of 100 consecutive words of data starting at offset address 300H in memory with another block of memory locations starting at offset address 600H. Assume both block at the same data segment F000H.